

Appendix A

```
package analysis;

5
import acme.*;
import java.util.*;
import java.io.*;
import java.awt.*;
10 import java.awt.event.*;
import javax.swing.*;

////////////////////////////////////
public class Analysis {
15
    // Temp for standalone analysis project. rundatastream.java
    public final static short TEMP = 7, OPTICS = 1 * 1024;

    public final static int NORMAL = 0, RAW = 1, DERIV1 = 2, DERIV2 = 3,
20     DERIV1RAW = 4, DERIV2RAW = 5, MELT_OPTICS = 6,
    MELT_TEMPERATURE = 7, MELT_DERIV1 = 8;
    public final static int MAX_CYCLES = 100;
    public final static int MAX_DYES = 4;
    public final static int MAX_SITES = 96;
25
    // Results
    public final static int PASS = 0;
    public final static int FAIL = 1;
    public final static int NO_RESULT = 2;    // eg, passive dye
30 public final static int ND = 3;           // Not Determined, IC invalid

    // Dye Types
    public final static int UNUSED = 0;
    public final static int ASSAY = 1;
35 public final static int INTERNAL_CONTROL = 2;
    public final static int QIC = 3;
    public final static int PASSIVE = 4;    // Historical but needed
    public final static int UNKNOWN = 5;    // Qual. Find conc. for this dye
    public final static int STANDARD = 6;    // Qual. Dye with known conc.
40

    // Site Designation
    public final static int SITE_UNKNOWN = 0;
    public final static int SITE_STANDARD = 1;

45 // Data to use
```

```

public final static int PRIMARY = 0;
public final static int D2 = 1;          // 2nd Derivative

// Analysis Type
5 public final static int QUALITATIVE = 0;
public final static int QUANTITATIVE = 1;

// Threshold mode
public final static int AUTO_THRESH = 0;
10 public final static int MAN_THRESH = 1;

public static boolean annotate = false;

// Setup, results...
15 Site site[];

private int analysisType;

// Num Sites
20 private int numSites;

// One per dye, site independent
// Primary: 0; 2D: 1
int dataType[] = new int[MAX_DYES];
25

// Following used for the standards curve, prakash 1/25/00
double dyeSlope[] = new double[MAX_DYES]; // m: mx+b
double dyeOffset[] = new double[MAX_DYES]; // b: mx+b
double linCC[] = new double[MAX_DYES];
30

// standardsLine[0-3][2]
// Each point is defined by (cycle, logb10(concentration))
public StdElement standardsLine[][] = new StdElement[MAX_DYES][2];
public static int stdChannel = 0;
35

// IC used: T, IC not used:F
private boolean useIC;
private int icDye;

// QIC used: T, QIC not used:F
40 private boolean useQIC;
private int qicDye;

// Threshold Mode (1 per dye)
45 private int threshMode[] = new int[MAX_DYES];

```

```

// Valid Cycle Number Range for all dyes
private float validMinCycle[] = new float[MAX_DYES];
private float validMaxCycle[] = new float[MAX_DYES];

5 // Cycle Number for noise sub and 3 sigma calculation.
boolean noise;
int baselineStartCycle[] = new int[MAX_DYES];
int baselineEndCycle[] = new int[MAX_DYES];

10 // StdDev baseline for auto threshold detect. User entered,
// one per dye.
private double stdDevBaseLine[] = new double[MAX_DYES];

// The Max stdDev for a given dye. one per dye
15 private float maxStdDev[] = new float[MAX_DYES];

// This is set to true only if all sites have a valid
// stdDev. Than only can you calculate the max.
private boolean maxStdDevValid[] = new boolean[MAX_DYES];

20 // BoxCar Averaging
private boolean boxCar;
private int boxCarWidth; // Note Min Value = 2

25 // Quantitative Analysis
public StdElement qtArr[][] = new StdElement[MAX_DYES][1];

// //////////////////////////////////////
30 // Keeps current settings, resets Data (and all calculated values from data)
// //////////////////////////////////////
public void resetData() {

    for(int s = 0; s < numSites; s++) {
35         site[s].cycle = 0;
        site[s].control = false;
        site[s].meltPoints = 0;

        for(int d = 0; d < MAX_DYES; d++) {
40             site[s].dye[d].tValid = false;
            site[s].dye[d].tCycle = 0f;
            site[s].dye[d].stdDevValid = false;
            site[s].dye[d].slope = 0.;
            site[s].dye[d].offset = 0.;
45             site[s].noiseValid[d] = false;
        }
    }

```

```

    }

    // qtArr = null;
    StdElement a[] = new StdElement[1];
5
    a[0] = new StdElement();

    // Site independent
    for(int d = 0; d < MAX_DYES; d++) {
10
        maxStdDev[d] = 0f;
        maxStdDevValid[d] = false;

        qtArr[d] = null;
        qtArr[d] = a;    // Reset Quantation

15
        standardsLine[d][0] = new StdElement();
        standardsLine[d][1] = new StdElement();
        dyeSlope[d] = 0.;
        dyeOffset[d] = 0.;
20
        linCC[d] = 0.;
    }
}

25
// //////////////////////////////////////
// Keeps current optics data, redoes all calculations. Eg. may be called
// after changing Threshold mode from manual to auto.
// //////////////////////////////////////
public void recalc() {
30
    int s, cy;

    //System.out.println("Analysis.recalc()");

    int c[] = new int[numSites];
35
    int meltCount[] = new int[numSites];

    for(s = 0; s < numSites; s++) {
        c[s] = site[s].cycle;
        meltCount[s] = site[s].meltPoints;
40
    }

    resetData();

    for(cy = 0; cy < c[0]; cy++) {
45
        for(s = 0; s < numSites; s++) {
            addCycle(s, site[s].dye[0].rOptic[cy], site[s].dye[1].rOptic[cy],

```

```

        site[s].dye[2].rOptic[cy], site[s].dye[3].rOptic[cy]);
    }
}
5
// //////////////////////////////////////
public void setNumSites(int s) {
    if(s <= 0) {
10        return;
    }

    if(s < numSites) {
        for(int i = s; i < numSites; i++) {
15            site[i] = null;
        }
    }
    numSites = s;
20 }

// //////////////////////////////////////
public void addCycle(int s, short op0, short op1, short op2, short op3) {
    int c = site[s].cycle;
25

    //System.out.println("addCycle Site " + s + " Op0 " + op0);

    site[s].dye[0].rOptic[c] = op0;
    site[s].dye[1].rOptic[c] = op1;
30    site[s].dye[2].rOptic[c] = op2;
    site[s].dye[3].rOptic[c] = op3;

    site[s].dye[0].pOptic[c] = op0;
    site[s].dye[1].pOptic[c] = op1;
35    site[s].dye[2].pOptic[c] = op2;
    site[s].dye[3].pOptic[c] = op3;

    processData(s);

40    ++site[s].cycle;
}

// //////////////////////////////////////
public void addMelt(int s, short secs, int type, short value) {
45

```

```

    //System.out.println("addMelt Site " + s + " sec " + secs + " type " + type + "
value " + value);

```

```

    site[s].meltPoints = secs;

```

```

    switch(type) {
    //case RunDataStream.OPTICS:
    case OPTICS:

```

```

        site[s].mOptic.set(secs, value);
        site[s].updateMeltDeriv();
        break;

```

```

    //case RunDataStream.TEMP:
    case TEMP:

```

```

        site[s].mTemp.set(secs, ((float)value / 100f));
        break;
    }
}

```

```

// //////////////////////////////////////

```

```

// 0=Ql, 1=Qn

```

```

public void setAnalysisType(int a) {
    analysisType = a;
}

```

```

// //////////////////////////////////////

```

```

// To Manually set Threshold limit

```

```

// Call this once per dye
public void setTLimit(int d, float tl) {
    for(int s = 0; s < numSites; s++) {
        site[s].dye[d].tLimit = tl;
    }
}

```

```

// //////////////////////////////////////

```

```

// For testing quantation only.

```

```

// Call this once per dye
private void setTCycle(int s, int d, float tc) {
    site[s].dye[d].tCycle = tc;
    site[s].dye[d].tValid = true;
}

```



```

        break;

        case QIC:
5         for(int si = 0; si < numSites; si++) {
            site[si].dye[d].dyeUsage = du;
        }

        useQIC = true;
10        qicDye = d;

        break;
    }
}

15

// //////////////////////////////////////
// d=Dye, sd = standard dev. Set by User
public void setStdDevbaseline(int d, double sd) {
20     stdDevBaseLine[d] = sd;
}

// //////////////////////////////////////
// IC and Qic
25 public void setICCycle(int d, int min, int max) {
    validMinCycle[d] = (float)min;
    validMaxCycle[d] = (float)max;
}

30

// //////////////////////////////////////
public void setNoiseSubtraction(boolean flag) {
35     noise = flag;
}

// //////////////////////////////////////
public void setBaselineCycle(int dye, int start, int end) {
40     baselineStartCycle[dye] = start;
    baselineEndCycle[dye] = end;
}

45 // //////////////////////////////////////
public void setBoxCarAvg(boolean flag, int width) {

```



```

        boxCar = flag;
        boxCarWidth = width;
    }

5
    // //////////////////////////////////////
    // Get Thresholds
    public float getTLimit(int s, int d) {
        //System.out.println("Analysis: getTLimit() " + site[s].dye[d].tLimit );
10    return site[s].dye[d].tLimit;
    }

    // //////////////////////////////////////
15    public float getTCycle(int s, int d) {
        if (site[s].dye[d].tCycle < validMinCycle[d] || site[s].dye[d].tCycle >
        validMaxCycle[d])
            return 0f;
        else
20            return site[s].dye[d].tCycle;
    }

    // //////////////////////////////////////
25    public float getQICTCycle(int s, int d) {

        int qicDye = getQICDye();
        float qicTCycle = getTCycle(s, qicDye);

30    if (useQIC && (qicTCycle > 0f)) {
        if (d == qicDye) return qicTCycle;
        return (getTCycle(s,d) / qicTCycle);
    }
    else
35    return 0f;
    }

    // //////////////////////////////////////
40    public boolean getTValid(int s, int d) {
        return site[s].dye[d].tValid;
    }

    // //////////////////////////////////////
45    public final double log10(double a) {
        if(a > 0.) {

```

```

        return (Math.log(a) / Math.log(10.));
    }
    else {
        return -9.5;
5    }
}

// //////////////////////////////////////
10 public final double log10(float a) {
    if(a > 0.) {
        return (Math.log((double) a) / Math.log(10.));
    }
    else {
15     return -9.5;
    }
}

20 // //////////////////////////////////////
// Get Results
// //////////////////////////////////////
public int getQLResult(int s, int d) {

25     int du = site[s].dye[d].dyeUsage;

    // Update IC
    if(useIC &&!site[s].control) {
        updatelC(s);
30    }

    if(du == UNUSED || du == PASSIVE) {
        site[s].dye[d].qlResult = NO_RESULT;
35    }
    else if(useIC) {
        if(site[s].control) {
            site[s].dye[d].qlResult = site[s].dye[d].tValid ? PASS : FAIL;
40        }
        else {
            site[s].dye[d].qlResult = ND;
        }
    }
45    else {
        site[s].dye[d].qlResult = site[s].dye[d].tValid ? PASS : FAIL;
    }
}

```

```

    }

    return site[s].dye[d].qlResult;
}

```

5

```

// //////////////////////////////////////
// Update Internal Control Status
void updateIC(int s) {

```

10

```

    if(site[s].dye[icDye].tValid) {

        // Also make sure it happened in the specified range
        if((site[s].dye[icDye].tCycle >= validMinCycle[icDye]) &&
15         (site[s].dye[icDye].tCycle <= validMaxCycle[icDye])) {
            site[s].control = true;
        }
        else {

```

20

```

            // Although .tValid, not in the range
            site[s].control = false;
        }
    }
    else {

```

25

```

        site[s].control = false;
    }
}

```

30

```

// //////////////////////////////////////
// Update Linear Correlation Coefficient
// //////////////////////////////////////
void updateCC(int d) {

```

35

```

    double yt, xt;
    double syy = 0., sxy = 0., sxx = 0., ay = 0., ax = 0.;

```

40

```

    if(qtArr[d].length < 2) {
        linCC[d] = 0.;
    }
    return;
}

```

45

```

    for(int j = 0; j < qtArr[d].length; j++) {
        ax += qtArr[d][j].conc;
        ay += qtArr[d][j].avgTCycle;
    }
}

```

```

    }

    ax /= qtArr[d].length;
    ay /= qtArr[d].length;
5
    for(int j = 0; j < qtArr[d].length; j++) {
        xt = qtArr[d][j].conc - ax;
        yt = qtArr[d][j].avgTCycle - ay;
        sxx += xt * xt;
10        syy += yt * yt;
        sxy += xt * yt;
    }

    linCC[d] = sxy / (Math.sqrt(sxx * syy));
15    linCC[d] *= linCC[d];
}

// //////////////////////////////////////
20 // 0. Check for unknown & thresh.
// 1. Check IC
// 2. Check QIC
// 3. Check for at least 2 data points in this qtArr
// 4. Check for unknown to be within knowns
25 // 5. Sort qtArr and Return unknown conc. Move to addstandard...
// //////////////////////////////////////
public double getQTResult(int s, int d) {

    double m = 1.0;
30

    // 0. Check for unknown thresh.
    if(!site[s].dye[d].tValid || (site[s].dye[d].dyeUsage != UNKNOWN)) {
        return 0.;
    }
35

    // 1. Check IC
    if(useIC) {
        if(!site[s].dye[icDye].tValid) {
            return 0.;
40        }
    }

    // 2. Check QIC
    // todo prakash.
45 // Should wait for all thresholds/site before constructing qtArr.
    if(useQIC) {

```



```

        break;
    }
}
}
5 while(!done);
}

// //////////////////////////////////////
10 // Sort the elements in the Melt Peaks Array.
void sort(MeltElement meltElementsArray[]) {

    boolean done;
    MeltElement me = new MeltElement();

15 //Debug.log ("sort: MeltElement array with " + meltElementsArray.length);
    if(meltElementsArray.length < 2) {
        return;
    }

20 do {
    done = true;

    for(int j = 0; j < (meltElementsArray.length - 1); j++) {
25 if(meltElementsArray[j].d1Peak > meltElementsArray[j + 1].d1Peak) {
        done = false;
        me = meltElementsArray[j];
        meltElementsArray[j] = meltElementsArray[j + 1];
        meltElementsArray[j + 1] = me;

30 break;
    }
}
} while(!done);
35 }

// //////////////////////////////////////
40 // Update data used for drawing the Line fit to standards.
//
// standardsLine is similar to qtArr[] but adds 2 points, one at
// conc +.5(log) and the other at conc -.5 (log).
// //////////////////////////////////////
45 void updateStandards(int d) {

```

```

int e = qtArr[d].length - 1;
double conc = qtArr[d][e].conc - .5;

standardsLine[d][0].conc = qtArr[d][0].conc + .5;
5 standardsLine[d][0].avgTCycle = (standardsLine[d][0].conc - dyeOffset[d])
    / dyeSlope[d];

if(conc > 0.) {
    standardsLine[d][1].conc = conc;
10 standardsLine[d][1].avgTCycle = (conc - dyeOffset[d]) / dyeSlope[d];
}
else {
    standardsLine[d][1].conc = 0.;
    standardsLine[d][1].avgTCycle = (-1 * dyeOffset[d] / dyeSlope[d]);
15 }
}

// //////////////////////////////////////
20 // Get Control Result (Pass/Fail)
// //////////////////////////////////////
public boolean getControl(int s, int d) {
    return site[s].control;
25 }

// //////////////////////////////////////
public float getConc(int s, int d) {
    return site[s].dye[d].conc;
30 }

// //////////////////////////////////////
public int getDyeUsage(int s, int d) {
35 return site[s].dye[d].dyeUsage;
}

// //////////////////////////////////////
40 public double getDyeSlope() {
    return dyeSlope[stdChannel];
}

45 // //////////////////////////////////////
public double getDyeOffset() {

```

```

    return dyeOffset[stdChannel];
}

```

```

5 ///////////////////////////////////////////////////////////////////

```

```

// Linear Correlation Coefficient

```

```

public double getCC() {
    updateCC(stdChannel);

```

```

10    return linCC[stdChannel];
}

```

```

///////////////////////////////////////////////////////////////////

```

```

15 public float getAnaData(int dataType, int s, int d, int c) {

```

```

    float retVal = 0f;

```

```

    if (c < 0) c=0;

```

```

20

```

```

    switch(dataType) {

```

```

        case NORMAL:

```

```

            if (c >=site[s].cycle) c=site[s].cycle - 1;

```

```

25

```

```

            if(d < 4 && d >= 0) {

```

```

                retVal = site[s].dye[d].pOptic[c];

```

```

            }

```

```

            break;

```

```

30

```

```

        case DERIV1:

```

```

            break;

```

```

        case DERIV2:

```

```

            if (c >=site[s].cycle) c=site[s].cycle - 1;

```

```

35

```

```

            if(d < 4 && d >= 0) {

```

```

                retVal = site[s].dye[d].d2pOptic[c];

```

```

            }

```

```

            break;

```

```

40

```

```

        case MELT_DERIV1:

```

```

            if (c >=site[s].meltPoints) c=site[s].meltPoints - 1;

```

```

            if(c < site[s].meltPoints && c >= 0) {

```

```

                retVal = site[s].d1mOptic.get(c);

```

```

            }

```

```

45

```

```

            break;

```



```

case MELT_OPTICS:
    if (c >= site[s].meltPoints) c = site[s].meltPoints - 1;
    if (c < site[s].meltPoints && c >= 0) {
        retVal = site[s].mOptic.get(c);
5      }
        break;

case MELT_TEMPERATURE:
    if (c >= site[s].meltPoints) c = site[s].meltPoints - 1;
10    if (c < site[s].meltPoints && c >= 0) {
        retVal = site[s].mTemp.get(c);
    }
        break;
    }

15    return retVal;
}

20 // //////////////////////////////////////
public int getICDye() {
    return icDye;
}

25 // //////////////////////////////////////
public boolean iCEnabled() {
    return useIC;
}

30 // //////////////////////////////////////
// Returns the temp assoc. with the Melt Peak.
public double getMeltTemp(int s, int index) {
    return site[s].getMeltTemp(index);
35 }

// //////////////////////////////////////
// Returns the Melt Limit. Peak value reported only when greater.
40 public double getMeltLimit(int s) {
    return site[s].meltPeakLimit;
}

45 // //////////////////////////////////////
// Returns the temp assoc. with the Melt Peak.

```

```

public int getMeltCount(int s) {
    if (s>0 && s<numSites)
        return site[s].getMeltPeakCount();
    else
5       return 0;
}

// //////////////////////////////////////
10 public int getQICDye() {
    return qicDye;
}

// //////////////////////////////////////
15 public boolean qicEnabled() {
    return useQIC;
}

// //////////////////////////////////////
20 public int getTMode(int d) {
    return threshMode[d];
}

// //////////////////////////////////////
25 int getICStartCycle() {
    return (int)validMinCycle[qicDye];
30 }

// //////////////////////////////////////
int getICEndCycle() {
35     return (int)validMaxCycle[qicDye];
}

// //////////////////////////////////////
40 void processData(int s) {

    if(boxCar) {
        boxCarAvg(s);
    }

45     if(noise) {

```

```

        removeNoise(s);
    }

    updateThresholds(s);

5    // Update qtArr's. Do quantation when results are requested.
    if(analysisType == QUANTITATIVE)
        updateQuantitative(s);
    }

10

    // //////////////////////////////////////
    // Apply this to raw Data
    void boxCarAvg(int s) {

15        float sum;
        int i;

        if(site[s].cycle < 1) {
20            return;
        }

        if(site[s].cycle + 1 >= boxCarWidth && boxCarWidth > 1) {

25            for(int d = 0; d < MAX_DYES; d++) {
                sum = 0f;

                for(i = (site[s].cycle + 1 - boxCarWidth); i < site[s].cycle + 1; i++) {
30                    sum += site[s].dye[d].rOptic[i];
                }

                site[s].dye[d].pOptic[site[s].cycle] = sum / boxCarWidth;
            }
        }

35    }

    // //////////////////////////////////////
    void removeNoise(int s) {

40        int c = site[s].cycle;
        float temp;

        for(int d = 0; d < MAX_DYES; d++) {
45            if(c >= (baselineEndCycle[d] - 1)) {

```

```

    if(site[s].noiseValid[d]) {
        site[s].dye[d].pOptic[c] -= (site[s].dye[d].slope * c + site[s].dye[d].offset);
        site[s].dye[d].pOptic[c] -= site[s].dye[d].noiseAvg;
5
        //if (s==0 && d==0) {
        //    Logger.log("Cycle "+c+ " slope "+site[s].dye[d].slope +
        //        " offset " + site[s].dye[d].offset + " pOptic " + site[s].dye[d].pOptic[c]);
        //}
10    }
    else {
        temp = 0f;

        // Calculate Average noise
15        baselineStartCycle[d] = (baselineStartCycle[d] < 1) ? 1 :
baselineStartCycle[d];

        site[s].dye[d].slope = 0.;
        site[s].dye[d].offset = 0.;
20
        site[s].dye[d].leastSquaresLineFit(baselineStartCycle[d]-1,
baselineEndCycle[d]-1);

        for(int i = 0; i <= (baselineEndCycle[d] - 1); i++) {
25            site[s].dye[d].pOptic[i] -= (site[s].dye[d].slope * i + site[s].dye[d].offset);
        }

        for(int i=baselineStartCycle[d]-1; i<=baselineEndCycle[d]-1; i++) {
            temp = temp + site[s].dye[d].pOptic[i];
30        }

        site[s].dye[d].noiseAvg = temp / (baselineEndCycle[d] -
baselineStartCycle[d] + 1);

35        // Remove noise
        for(int i=0; i <= (baselineEndCycle[d]-1); i++) {
            site[s].dye[d].pOptic[i] -= site[s].dye[d].noiseAvg;
        }
        site[s].noiseValid[d] = true;
40    }
    }
    }
    }
45
    // //////////////////////////////////////

```

```

void updateThresholds(int s) {
    for(int d = 0; d < MAX_DYES; d++) {

```

```

5    // Update Derivative
    update2D(s, d);

```

```

    if(dataType[d] == PRIMARY) {
        if(threshMode[d] == MAN_THRESH) {
10        updateThreshPDMan(s, d);
        }
        else {
            updateThreshPDAuto(s, d);
        }
15    }
    else {
        if(threshMode[d] == MAN_THRESH) {
            updateThresh2DMan(s, d);
        }
20        else {
            updateThresh2DAuto(s, d);
        }
    }
25 }
}

```

```

// //////////////////////////////////////
int updateThreshPDMan(int s, int d) {

```

```

30    int c = site[s].cycle;
    int du = site[s].dye[d].dyeUsage;

    if(du == UNUSED || du == PASSIVE) {
35        return 0;
    }

```

```

    if(noise) {
        if(c <= baselineEndCycle[d]) {
40            return 0;
        }
    }

```

```

    if(!site[s].dye[d].tValid) {
45        if(site[s].dye[d].pOptic[c] >= site[s].dye[d].tLimit) {

```

```

        // Optic exceeded limit, calculate cycle
        if(c >= 1) {
            site[s].dye[d].tValid = true;

5           LinearFit l;

            l = new LinearFit(c - 1, site[s].dye[d].pOptic[c - 1], c,
                            site[s].dye[d].pOptic[c]);

10          // zero based
            site[s].dye[d].tCycle = l.fitY(site[s].dye[d].tLimit) + 1f;
        }
    }
15    return 0;
}

// //////////////////////////////////////
20    // When not to find the Threshold crossing:
    //
    // 1. Unused Dye
    // 2. Passive dye
    // 3. Already found (.tValid)
25    // 4. Not enough cycles (2D)
    // 5. All dyes don't have valid stdDev Auto
    // //////////////////////////////////////
    int updateThreshPDAuto(int s, int d) {

30        int c = site[s].cycle;
        float sum, temp;
        int du = site[s].dye[d].dyeUsage;

        if(du == UNUSED || du == PASSIVE) {
35            return 0;
        }

        if(c <= baselineEndCycle[d]) {
40            return 0;
        }

        if(maxStdDevValid[d] &&!site[s].dye[d].tValid) {

            // Look for signal crossing
45            if(site[s].dye[d].pOptic[c] > site[s].dye[d].tLimit) {

```

LinearFit l;

l = new LinearFit(c - 1, site[s].dye[d].pOptic[c - 1], c, site[s].dye[d].pOptic[c]);

```
5    // Add one to match graph
    site[s].dye[d].tCycle = l.fitY(site[s].dye[d].tLimit) + 1.0f;
    site[s].dye[d].tValid = true;
    }
```

```
10   else if(!maxStdDevValid[d] &&!site[s].dye[d].tValid) {
```

```
    // If enough data, calculate stdDev
    // No need to check crossing yet.
    if(c >= baselineEndCycle[d]) {
```

```
15     if((baselineEndCycle[d] - baselineStartCycle[d]) > 1) {
```

```
        // mean
        sum = 0f;
```

```
20     for(c = (baselineStartCycle[d] - 1); c <= (baselineEndCycle[d] - 1); c++) {
        sum = sum + site[s].dye[d].pOptic[c];
    }
```

```
25     site[s].dye[d].mean = sum / (baselineEndCycle[d] - baselineStartCycle[d] +
    1);
```

```
    // stdDev
    sum = 0f;
```

```
30     for(c = (baselineStartCycle[d] - 1); c <= (baselineEndCycle[d] - 1); c++) {
        temp = site[s].dye[d].pOptic[c] - site[s].dye[d].mean;
        sum = sum + temp * temp;
    }
```

```
35     site[s].dye[d].stdDev = (float) Math.sqrt(sum / (baselineEndCycle[d] -
baselineStartCycle[d]));
    site[s].dye[d].stdDevValid = true;
```

```
    setMaxStdDev(d);
40   }
    }
}
```

```
    return 0;
45 }
```

```

// //////////////////////////////////////
// This function calculates the Cycle Threshold for Primary Data with
// a manual threshold limit set by the user.
5 // //////////////////////////////////////
int updateThresh2DMan(int s, int d) {

    int du = site[s].dye[d].dyeUsage;

10 // Because the calculation for D2 is lagging 2 cycles back.
    int c = site[s].cycle - 2;

    if(du == UNUSED || du == PASSIVE) {
        return 0;
15 }

    if(c < 6) {
        return 0;
    }

20 if(noise) {
    if(c <= baselineEndCycle[d]) {
        return 0;
    }
25 }

    // Look for peak
    // When c == 6, Possible valid D2's are at c2(c-4), c3(c-3), c4(c-2)
    if((site[s].dye[d].d2pOptic[c - 3] > site[s].dye[d].d2pOptic[c - 4]) &&
30 (site[s].dye[d].d2pOptic[c - 3] >= site[s].dye[d].d2pOptic[c - 2])) {

        PeakFinder peakFinder = new PeakFinder((float) (c - 4),
        site[s].dye[d].d2pOptic[c - 4],
        (float) (c - 3), site[s].dye[d].d2pOptic[c - 3], (float) (c - 2),
35 site[s].dye[d].d2pOptic[c - 2]);

        // Look for signal crossing
        if(peakFinder.peak > site[s].dye[d].tLimit) {

40 // peak exceeded limit, calculate cycle
        // Note: peak is 3 cycles back from here
        if(site[s].dye[d].tValid) {

            if (site[s].dye[d].tCycle < peakFinder.cycle + 1.0f) {
45 site[s].dye[d].tCycle = peakFinder.cycle + 1.0f;
            }
        }
    }
}

```



```

    }
    else {
        site[s].dye[d].tValid = true;
5       site[s].dye[d].tCycle = peakFinder.cycle + 1.0f;
    }
}
}
return 0;
10 }

```

```

// //////////////////////////////////////
// //////////////////////////////////////
15 int updateThresh2DAuto(int s, int d) {

    int du = site[s].dye[d].dyeUsage;
    float sum, temp;
    int cy;

20     // Because the calculation for D2 is lagging 2 cycles back.
    int c = site[s].cycle - 2;

    if(du == UNUSED || du == PASSIVE) {
25         return 0;
    }

    if(c < 6) {
        return 0;
30     }

    if(c <= baselineEndCycle[d]) {
        return 0;
    }

35     if(maxStdDevValid[d]) {

        // Look for signal crossing, ie Look for peak
        // When c == 6, Possible valid D2's are at c2(c-4), c3(c-3), c4(c-2)
40         if(c < (baselineEndCycle[d] + 3)) {
            return 0;
        }

        if((site[s].dye[d].d2pOptic[c - 3] >= site[s].dye[d].d2pOptic[c - 4]) &&
45         (site[s].dye[d].d2pOptic[c - 3] > site[s].dye[d].d2pOptic[c - 2])) {

```

```

PeakFinder m = new PeakFinder((float) (c - 4), site[s].dye[d].d2pOptic[c - 4],
    (float) (c - 3), site[s].dye[d].d2pOptic[c - 3], (float) (c - 2),
    site[s].dye[d].d2pOptic[c - 2]);

```

```

5 // Look for signal crossing
  if(m.peak > site[s].dye[d].tLimit) {

```

```

    if (site[s].dye[d].tValid) {
      if (site[s].dye[d].tCycle < m.cycle + 1f) {
10      site[s].dye[d].tCycle = m.cycle + 1f;
      }
    }
  }

```

```

    else {
      // peak exceeded limit, calculate cycle
15      site[s].dye[d].tValid = true;
      site[s].dye[d].tCycle = m.cycle + 1f;
    }
  }
}

```

```

20 }
else if(!maxStdDevValid[d] &&!site[s].dye[d].tValid) {

```

```

    // If enough data, calculate stdDev
    // No need to check crossing yet.

```

```

25 if(c >= baselineEndCycle[d]) {
    if((baselineEndCycle[d] - baselineStartCycle[d]) > 1) {

```

```

        // mean
        sum = 0f;

```

```

30 for(c = (baselineStartCycle[d] - 1); c <= (baselineEndCycle[d] - 1); c++) {
    sum = sum + site[s].dye[d].d2pOptic[c];
  }

```

```

35 // Changed 1/12/00 as per SCR 129.
    // sum = sum + site[s].dye[d].pOptic[c];

```

```

    site[s].dye[d].mean = sum / (baselineEndCycle[d] - baselineStartCycle[d] +
40 1);

```

```

    // stdDev
    sum = 0f;

```

```

45 for(c = (baselineStartCycle[d] - 1); c <= (baselineEndCycle[d] - 1); c++) {

```

```

    // Changed 1/12/00 as per SCR 129.

```

```

        // temp = site[s].dye[d].pOptic[c] - site[s].dye[d].mean;

        temp = site[s].dye[d].d2pOptic[c] - site[s].dye[d].mean;
        sum = sum + temp * temp;
5      }

        site[s].dye[d].stdDev = (float) Math.sqrt(sum / (baselineEndCycle[d] -
baselineStartCycle[d]));
        site[s].dye[d].stdDevValid = true;
10      setMaxStdDev(d);
    }
}
}
15  return 0;
}

20  // //////////////////////////////////////
// Update 2nd Deriv for optic data
// //////////////////////////////////////
void update2D(int s, int d) {

25      int c = site[s].cycle;
      float mult = 6.25f;

      if (c<4)
          return;
30      // D2
      if(c < MAX_CYCLES - 1 && c > 2) {

          // n=3 thru n-2
35          /*
          //float mult = 5f;
          site[s].dye[d].d2pOptic[c - 2] = (site[s].dye[d].arD1Dye[c - 1] -
              site[s].dye[d].arD1Dye[c - 3]) / 2f * mult;
40          site[s].dye[d].d2pOptic[c - 1] = (site[s].dye[d].arD1Dye[c] -
              site[s].dye[d].arD1Dye[c - 2]) / 2f * mult;
          site[s].dye[d].d2pOptic[c] = (site[s].dye[d].arD1Dye[c] -
              site[s].dye[d].arD1Dye[c - 1]) * mult;
          */
45          site[s].dye[d].d2pOptic[c-2] = (site[s].dye[d].pOptic[c] -
              2f * site[s].dye[d].pOptic[c-2] +

```

```
site[s].dye[d].pOptic[c-4]) * mult;
```

```
site[s].dye[d].d2pOptic[c-1] = (2f * site[s].dye[d].pOptic[c] -  
3f * site[s].dye[d].pOptic[c-1] +  
5 site[s].dye[d].pOptic[c-3]) * mult;
```

```
site[s].dye[d].d2pOptic[c] = (site[s].dye[d].pOptic[c] -  
2f * site[s].dye[d].pOptic[c-1] +  
site[s].dye[d].pOptic[c-2]) * 2 * mult;
```

```
10 }  
else {  
    site[s].dye[d].d2pOptic[c] = 0f;  
}  
}
```

```
15  
  
// ///////////////////////////////////////  
// Update qtArr's (1 per dye - site independent).  
// Only if std: only with valid thresh  
20 // ///////////////////////////////////////  
void updateQuantitative(int s) {
```

```
    for(int d = 0; d < MAX_DYES; d++) {  
        if(site[s].dye[d].dyeUsage == STANDARD) {  
25            // if(site[s].dye[d].tValid) {  
  
                if( (useQIC && (getTCycle(s, qicDye) > 0f)) || getTCycle(s, d) > 0f ) {  
                    addStandard(s, d);  
                    //updateStandards(d);  
30                    LeastSquares ls = new LeastSquares(qtArr[d], d);  
                    dyeSlope[d] = ls.getSlope();  
                    dyeOffset[d] = ls.getOffset();  
                    updateStandards(d);  
                }  
            }  
35        }  
    }  
}
```

```
40 // ///////////////////////////////////////  
//  
// Add a stdElement to the qlArr if appropriate.  
// If QIC used - valid  
// If IC used - valid  
45 // Sort if more than 1 element  
// ///////////////////////////////////////  

```

```

int addStandard(int s, int d) {
    int i;
    float tCycle;

5    if(!site[s].dye[d].tValid || getTCycle(s,d) <= 0f ) {
        return 0;
    }

    if(site[s].dye[d].conc < 10E-5f) {
10        return 0;
    }

    if (useQIC) {
        tCycle = getQICTCycle(s,d);
15    }
    else {
        tCycle = getTCycle(s,d);
    }

20    if (qtArr[d][0].conc < -9) {
        // Initialise
        qtArr[d][0].conc = log10(site[s].dye[d].conc);
        qtArr[d][0].avgTCycle = tCycle;
        qtArr[d][0].nElements = 1;
25        return 0;
    }
    else {

        // Look for conc in array
30        for(i = 0; i < qtArr[d].length; i++) {

            if(Math.abs(qtArr[d][i].conc - log10(site[s].dye[d].conc)) < .05) {
                qtArr[d][i].avgTCycle = ((qtArr[d][i].avgTCycle * qtArr[d][i].nElements) +
                    tCycle) / (qtArr[d][i].nElements + 1);
35                qtArr[d][i].nElements += 1;

                // May need to be resorted
                if(qtArr[d].length > 1) {
                    sort(qtArr[d]);
40                }

                return 0;
            }
        }
45        // Conc not found, add new element to array

```



```

// All sites have stdDevValid for dye d,
// Calculate Threshold limits
for(s = 0; s < numSites; s++) {
5   site[s].dye[d].tLimit = (float)(stdDevBaseLine[d] * maxStdDev[d]);
   //System.out.println("stdDevBaseLine[d] " + stdDevBaseLine[d] +
   // "maxStdDev[d] " + maxStdDev[d] +
   // " setMaxStdDev " + site[s].dye[d].tLimit );
   }
10  }
   }

// //////////////////////////////////////
15  public Analysis() {
   this(MAX_SITES);
   }

   public Analysis(int ns) {
20     numSites = ns;

   site = new Site[numSites];

25   for(int i = 0; i < numSites; i++) {
   site[i] = new Analysis.Site();
   }

   analysisType = QUALITATIVE;
30   useQIC = false;
   qicDye = 0;
   useIC = false;
   icDye = 0;
35   boxCar = false;
   boxCarWidth = 0;

   // Default to match noise sub with primary data.
40   // noise = false;

   for(int i = 0; i < MAX_DYES; i++) {
   threshMode[i] = AUTO_THRESH;
   stdDevBaseLine[i] = 5f;
45   maxStdDev[i] = 0f;
   maxStdDevValid[i] = false;

```

```

    dataType[i] = PRIMARY;
    qtArr[i][0] = new StdElement();
    baselineStartCycle[i] = 3;
    baselineEndCycle[i] = 8;
5
    // Standards Curve, prakash 1/25/00
    standardsLine[i][0] = new StdElement();
    standardsLine[i][1] = new StdElement();

10    // Optics must cross threshold in this range
    validMinCycle[i] = 3f;
    validMaxCycle[i] = 60f;
    }
    }

15
    // //////////////////////////////////////
    class Site {

20        Dye dye[] = new Dye[MAX_DYES];

        // Melt Peak Analysis
        private Array.Short mOptic = new Array.Short(32);
        private Array.Float mTemp = new Array.Float(32);
25        private Array.Float d1mOptic = new Array.Float(32);
        private MeltElement mPeaks[] = new MeltElement[1];

        // Possible to set per site in future.
        private double meltPeakLimit = 10.;

30        // Melt peaks processed
        private boolean meltPeaksValid;

        // Current Cycle Number
35        int cycle;

        // Number of MeltData points
        private int meltPoints;

40        // IC/QIC passed:T; failed:F
        boolean control;

        // Noise
        boolean noiseValid[] = new boolean [MAX_DYES];

45        Site() {

```



```

// Initialise dyes
for(int i = 0; i < MAX_DYES; i++) {
    dye[i] = new Dye();
    noiseValid[i] = false;
}

cycle = 0;
meltPoints = 0;
meltPeaksValid = false;
control = false;
mPeaks[0] = new MeltElement();
}

private void updateMeltDeriv() {

    meltPeaksValid = false;

    if(meltPoints < 1) {
        d1mOptic.set(0, 0f);
    }
    else if(meltPoints == 1) {
        d1mOptic.set(1, (mOptic.get(1) - mOptic.get(0)) * -5f);
    }
    else {
        // Recalc the 2nd last value, and the last value
        d1mOptic.set(meltPoints-1, (mOptic.get(meltPoints) -
mOptic.get(meltPoints-2)) / 2f * -5f);
        d1mOptic.set(meltPoints, (mOptic.get(meltPoints) -
30 mOptic.get(meltPoints-1)) * -5f);
    }
}

// Return number of Melt Peaks detected.
private int getMeltPeakCount() {
    if (!meltPeaksValid)
        detectMeltPeaks();
    return (mPeaks[0].temp < 0.) ? 0 : mPeaks.length;
}

// Return number of Melt Temp Associated with Peak.
private double getMeltTemp(int index) {
    if (index < getMeltPeakCount())
        return mPeaks[index].temp;
45     else
        return 0f;
}

```

```

    }

    // Find all peaks in 1st Deriv of Melt Optic
    private void detectMeltPeaks() {
5
        if (meltPoints < 2) return;

        if (!meltPeaksValid) {
            meltPeaksValid = true;
10
            mPeaks = new MeltElement[1];
            mPeaks[0] = new MeltElement();
            // Debug.log("detectMP, length " + mPeaks.length);

            for (int i=1; i<meltPoints-1; i++) {
15
                if ( ( d1mOptic.get(i) > d1mOptic.get(i-1) ) &&
                    ( d1mOptic.get(i) >= d1mOptic.get(i+1) ) ) {

                    PeakFinder peakFinder = new PeakFinder((float)(i-1),
20
                    (float)d1mOptic.get(i-1),
                    (float)i, (float)d1mOptic.get(i), (float)(i+1),
                    (float)d1mOptic.get(i+1));

                    // Look for signal crossing
25
                    if(peakFinder.peak > meltPeakLimit) {

                        if (mPeaks[0].temp < 0.) {
                            mPeaks[0].d1Peak = peakFinder.peak;
                            mPeaks[0].temp = mTemp.get(0) + peakFinder.cycle; // Temp,
30
                            in this case.
                        }
                        else {
                            MeltElement tempA[] = new MeltElement[mPeaks.length+1];

                            // Initialise tempA
35
                            for(int j = 0; j < tempA.length; j++) {
                                tempA[j] = new MeltElement();
                            }

                            System.arraycopy(mPeaks, 0, tempA, 0, mPeaks.length);

                            tempA[tempA.length-1].d1Peak = peakFinder.peak;
                            tempA[tempA.length-1].temp = mTemp.get(0) +
45
                            peakFinder.cycle; // Temp, in this case.
                            mPeaks = tempA;
                        }
                    }
                }
            }
        }
    }

```

```

    }
    }
    }
5    //Debug.log(" detectMeltPeaks() mPeaks.length " + mPeaks.length);
    if (mPeaks.length > 1)
        sort(mPeaks);
    }
10 }

```

```

// //////////////////////////////////////
class Dye {

```

```

15    // Data Arrays
    short rOptic[] = new short[MAX_CYCLES];
    float pOptic[] = new float[MAX_CYCLES];

    // 2nd derivative
20    float d2pOptic[] = new float[MAX_CYCLES];

    // Threshold limit
    float tLimit;
    float tCycle;
25

    // Indicates if signal crossed the Threshold Limit
    boolean tValid;

    // Qualitative Result
30    int qlResult;

    // IC, QIC, Unused, ...
    int dyeUsage;

35    // true = Std; false = Unkn
    boolean std;

    // Dye Concentration
    float conc;
40

    // Background Noise Value
    float noiseAvg;

    // Std Dev, Mean calculated. one per dye per site
45    boolean stdDevValid;
    float stdDev;

```

float mean;

// For slope removal. One per dye per site

double slope;

double offset;

Dye() {

// Initialise arrays

for(int i = 0; i < MAX_CYCLES; i++) {

 rOptic[i] = 0;

 pOptic[i] = 0f;

 d2pOptic[i] = 0f;

}

// Default Man Threshold, dyeUsage, tValid

qlResult = 0;

tLimit = 200f;

tCycle = 0f;

tValid = false;

dyeUsage = ASSAY;

std = false;

conc = 10E-6f;

noiseAvg = 0f;

stdDevValid = false;

stdDev = 0f;

mean = 0f;

slope = 0.;

offset = 0.;

}

void endPointLineFit(int start, int end) {

 slope = (pOptic[end] - pOptic[start]) / (double)(end - start);

 if ((slope * end) != 0.) {

 offset = pOptic[end] / (slope * end);

 }

 else {

 offset = 0.;

 }

}

void leastSquaresLineFit(int start, int end) {

 if ((end - start) < 2) {

 return;

```

    }
    LeastSquares ls = new LeastSquares(pOptic, start, end);
    slope = ls.getSlope();

5      if ((slope * end) != 0.) {
        offset = ls.getOffset();
      }
      else {
        offset = 0.;
10     }
  }
}

```

```

15  // //////////////////////////////////////
public class StdElement {
    public double conc;
    public double avgTCycle;
    int nElements;

20      StdElement() {
        conc = -10.;
        avgTCycle = 0.;
        nElements = 0;
25    }
}

```

```

// //////////////////////////////////////
public class MeltElement {
30    public double temp = -1.;
    public double d1Peak = -1.;
}

```

```

35  // //////////////////////////////////////
    // //////////////////////////////////////
public static void main(String args[]) {

40      int s, d, c, cy;
      Analysis a = new Analysis();

      // For reading data from Excel
      Vector vFam = new Vector(16);
45      vFam.setSize(16);
      Vector vTet = new Vector(16);

```

```

    vTet.setSize(16);
    Vector vTam = new Vector(16);
    vTam.setSize(16);
    Vector vRox = new Vector(16);
    vRox.setSize(16);

    // Analysis Type
    a.setAnalysisType(QUALITATIVE);
    //a.setAnalysisType(QUANTITATIVE);

    a.setNumSites(16);

    for (d=0; d<MAX_DYES; d++) {

        //a.setDataType(d, D2);          // Set Up Data Type
        a.setDataType(d, PRIMARY);

        a.threshMode[d] = AUTO_THRESH; // Set Thresh Mode
        //a.threshMode[d] = MAN_THRESH;

        a.stdDevBaseLine[d] = 5.;
    }

    // Set Threshold
    //a.setTLimit(0, 10f);
    //a.setTLimit(1, 10f);
    //a.setTLimit(2, 10f);
    //a.setTLimit(3, 10f);

    // Test BoxCar Avg
    a.setBoxCarAvg(true, 3);

    // Test QIC Dye
    a.setDyeUsage(0, 1, QIC);

    // Test Background Noise Subtraction
    a.setNoiseSubtraction(true);

    // Valid Min, Max Cycle defaults to 3, 60
    //a.setlCCycle(3, 30, 60);

    // Add Data Thresholds and cycle crossings are calculated as soon as
    // enough data has accumulated.

```

```

        try {
            BufferedReader in = new BufferedReader(new
FileReader("data5.csv"));

5           String str;

            // Throw away first 2 lines
            str = in.readLine();
            str = in.readLine();

10          while ((str = in.readLine()) != null) {
                //Debug.log(str.length()+" "+ str);
                StringTokenizer t = new StringTokenizer(str, ",");

15          for (int i=0; i<16; i++)
                if (t.hasMoreTokens())
                    vFam.setElementAt( (Integer.valueOf(t.nextToken())), i);

            for (int i=0; i<16; i++)
20          for (int i=0; i<16; i++)
                if (t.hasMoreTokens())
                    vTet.setElementAt((Integer.valueOf(t.nextToken() )), i );

            for (int i=0; i<16; i++)
                if (t.hasMoreTokens())
25          vTam.setElementAt((Integer.valueOf(t.nextToken() )), i );

            for (int i=0; i<16; i++)
                if (t.hasMoreTokens())
                    vRox.setElementAt((Integer.valueOf(t.nextToken() )), i );

30          for (s=0; s<16; s++) {

                Integer aa = (Integer)vFam.elementAt(s);
                Integer bb = (Integer)vTet.elementAt(s);
                Integer cc = (Integer)vTam.elementAt(s);
                Integer dd = (Integer)vRox.elementAt(s);

35          a.addCycle(s, aa.shortValue(), bb.shortValue(),
cc.shortValue(), dd.shortValue() );

40          // cy = a.site[s].cycle -1;
                //Debug.log("Main: Site " +s+ " Cycle " +cy+ " " +
a.site[s].dye[0].rOptic[cy]+
                // " "+a.site[s].dye[1].rOptic[cy]+
45          // " "+a.site[s].dye[2].rOptic[cy]+
                // " "+a.site[s].dye[3].rOptic[cy] );

```

```

        }
    }
}
5      catch(IOException e) {
        Debug.log("IOException");
    }

    // Set up Melt Inverse of FAM
10     for (s=0; s<16; s++) {
        for (short sec=0; sec<a.site[s].cycle; sec++) {
            //Debug.log ("Adding data to Melt " + sec + " " +
a.site[s].dye[1].rOptic[sec]);
            a.addMelt(s, sec, a.OPTICS, a.site[s].dye[1].rOptic[sec]);
15         a.addMelt(s, sec, a.TEMP, (short)(60+sec));
        }
    }

    /*
20     // Set UP for quantation.
    // 100
    a.setSiteType(0, SITE_STANDARD);
    a.setConc(0, 0, 100f);

25     a.setSiteType(1, SITE_STANDARD);
    a.setConc(1, 0, 100f);

    //1000
    a.setSiteType(3, SITE_STANDARD);
30     a.setConc(3, 0, 1000f);

    a.setSiteType(8, SITE_STANDARD);
    a.setConc(8, 0, 1000f);

35     //10
    a.setSiteType(14, SITE_STANDARD);
    a.setConc(14, 0, 10f);

    a.setSiteType(15, SITE_STANDARD);
40     a.setConc(15, 0, 10f);

    // Unknowns
    a.setSiteType(2, SITE_UNKNOWN);
    a.setSiteType(4, SITE_UNKNOWN);
45     a.setSiteType(5, SITE_UNKNOWN);
    a.setSiteType(6, SITE_UNKNOWN);

```



```

a.setSiteType(7, SITE_UNKNOWN);

for (int i=9; i<14; i++)
    a.setSiteType(i, SITE_UNKNOWN);
5  */

/*
// Force QIC Cycle for testing
for (int i=0; i<16; i++) {
10    a.setTCycle(i, 1, (float)(10+.1*i));
    //a.setTCycle(i, 1, 10f );
    a.site[i].dye[1].tValid = true;
}

15    for(int i=0; i<a.numSites; i++)
        a.updateQuantitative(i);
    */

    // (site, dye, data)
20    //a.dLog(7, 1, 1); // outputs threshold limits + Cycle num
    //a.dLog(7, 0, 0); // outputs data
    //a.dLog(7, 1, 2); // outputs raw + 2d
    //a.dLog(7, 0, 3); // outputs threshold limits + Cycle num
    //a.dLog(7, 0, 4); // outputs threshold limits + Cycle num + QIResult
25    //a.dLog(0, 0, 5); // outputs Tlimits + TCycle num + conc (dye, all
    sites)

    //a.dLog(0, 0, 6); // outputs qtArr for given dye
    //a.dLog(7, 1, 7); // outputs threshold limits + Cycle num + QIC
    Cycle numbers
30    //a.dLog(7, 1, 8); // Outputs melt data for given site.
    //a.dLog(7, 1, 9); // Outputs melt data peaks for given site.

    Debug.log("*****");
    Debug.log("data4.csv, primary w Man Thresh,
35    setNoiseSubtraction(true));
    Debug.log("setBoxCarAvg(true, 3) Quantitative ");
    Debug.log("*****");
    a.dLog(3, 0, 2);
    }
40

    //////////////////////////////////////
    //////////////////////////////////////
    // Used for unit testing
45    void dLog(int st, int dy, int data) {

```

```

int i, s, d, c;

switch (data) {
case 0:
5    // data
    Debug.log("dLog: pOptic 7,* - Cy 0-44");

    for (i=0; i<site[st].cycle; i++)
        Debug.log(" " + site[st].dye[0].pOptic[i] +
10        " " + site[st].dye[1].pOptic[i] +
        " " + site[st].dye[2].pOptic[i] +
        " " + site[st].dye[3].pOptic[i] );
        break;
    case 1:
15        // thresh Limits, Cycle Numbers
        for (s=0; s<numSites; s++)
            for (d=0; d<MAX_DYES; d++)
                Debug.log("Site " + s +
                " Dye " + d +
20                " Thresh " + getTLimit(s, d) +
                " Cycle " + getTCycle(s, d) );
                break;

        // Prints raw + 2d data for st, dy
25    case 2:
        for (c=0; c<site[st].cycle; c++)
            Debug.log("Site " + st +
            " Dye " + dy +
            " Cycle " + c +
30            " raw data " + site[st].dye[dy].rOptic[c] +
            " data " + site[st].dye[dy].pOptic[c] +
            " 2D " + site[st].dye[dy].d2pOptic[c] );
            break;

35    // Prints dy channel TCycles and TLimits
    case 3:
        for (s=0; s<numSites; s++)
            Debug.log("Site " + s +
            " Dye " + dy +
40            " Thresh Cycle " + getTCycle(s, dy) +
            " Thresh Limit " + getTLimit(s, dy)
            );
            break;

45    // Prints dy channel TCycles and TLimits and QI Results
    case 4:

```

```

5      for (s=0; s<numSites; s++)
        Debug.log("Site " + s +
          " Dye " + dy +
          " Thresh Cycle " + getTCycle(s, dy) +
          " Thresh Limit " + getTLimit(s, dy) +
          " Result " + getQLResult(s, dy)
        );
        break;

10     // Prints dy channel TCycles and Qn Results
    // for dye at all sites
    case 5:
    for (s=0; s<numSites; s++)
        if (useQIC) {
15            Debug.log("Site " + s +
                " Dye " + dy +
                " QIC Thresh Cycle " + getQICTCycle(s, dy) +
                " Result " + getQTRResult(s, dy)
            );
20        }
        else {
            Debug.log("Site " + s +
                " Dye " + dy +
                " Thresh Cycle " + getTCycle(s, dy) +
25            " Result " + getQTRResult(s, dy)
            );
        }
        break;

30     case 6:
        for (c=0; c<qtArr[0].length; c++)
            Debug.log(" qtArr[0] Len "+ qtArr[0].length +" conc "+
qtArr[0][c].conc+ " Avg cy "+ qtArr[0][c].avgTCycle);
        break;

35     // Prints dy channel TCycles and TLimits + QIC
    case 7:
    for (s=0; s<numSites; s++) {
        for (dy=0; dy<4; dy++) {
40            Debug.log("Site " + s +
                " Dye " + dy +
                " Thresh Cycle " + getTCycle(s, dy) +
                " QIC Thresh Cycle " + getQICTCycle(s, dy) +
                " Thresh Limit " + getTLimit(s, dy)
45            );
        }
    }

```

```

        }
        break;

        // Prints melt for given site
        case 8:
5       for (c=0; c<site[st].cycle; c++) {
            Debug.log("Site " + st +
                " sec " + c +
                " mOptic " + site[st].mOptic.get(c) +
10         " d1mOptic " + site[st].d1mOptic.get(c) +
                " Temp " + site[st].mTemp.get(c)
            );
        }
        break;

        // Prints melt Peaks for given site
        case 9:
15       for (c=0; c<site[st].getMeltPeakCount(); c++) {
            Debug.log("Site " + st +
                " MeltPoint " + c +
                " d1peak " + site[st].mPeaks[c].d1Peak +
20         " temp " + getMeltTemp(st, c)
            );
        }
25       break;
    }
}
}

```

```
// //////////////////////////////////////
// Least Squares Fit. Takes an array of points (x,y pairs) and calculates
// the slope and offset using the 'Least Squares Fit' method.
// //////////////////////////////////////
```

```
5  class LeastSquares {

    double sumX = 0.;
    double sumY = 0.;
    double sumXY = 0.;
10  double sumOfXSq = 0.;
    double sumXSquared = 0.;
    int arrayLen = 0;
    double slope = 0.;

15  LeastSquares() {};

    // Used for quantation.
    LeastSquares(Analysis.StdElement a[], int d) {

20      arrayLen = a.length;

        for(int i = 0; i < arrayLen; i++) {
            sumX += a[i].avgTCycle;
            sumY += a[i].conc;
25      sumXY += a[i].avgTCycle * a[i].conc;
            sumOfXSq += a[i].avgTCycle * a[i].avgTCycle;
        };
        sumXSquared = sumX * sumX;
    };

30

    // Used for removing background noise
```

LeastSquares(float optic[], int start, int end) {

arrayLen = end - start + 1;

5 for(int i = start; i < end+1; i++) {

 sumX += i;

 sumY += optic[i];

 sumXY += i * optic[i];

 sumOfXSq += i * i;

10 }

 sumXSquared = sumX * sumX;

};

double getSlope() {

15 if(Math.abs(sumOfXSq - sumXSquared / arrayLen) > 10E-10) {

 slope = (sumXY - (sumY * sumX / arrayLen)) /

 (sumOfXSq - (sumXSquared / arrayLen));

 }

 else {

20 slope = 0.;

 }

 return slope;

}

25 double getOffset(){

 return (sumY / arrayLen) - (slope * sumX / arrayLen);

 }

}

```
// //////////////////////////////////////
// This object takes 2 points (x,y) pairs and calculates the slope and
// offset. It returns the unknown (either x or y) using the equation
//  $y = mx + b$ .
```

```
5 // //////////////////////////////////////
```

```
class LinearFit {
```

```
    double m;
```

```
    double b;
```

```
10
```

```
    LinearFit() {};
```

```
    LinearFit(int x1, double y1, int x2, double y2) {
```

```
        m = 0.;
```

```
15
```

```
        b = 0.;
```

```
        if((x1 - x2) != 0) {
```

```
            m = (y1 - y2) / (x1 - x2);
```

```
            b = y1 - m * x1;
```

```
20
```

```
        }
```

```
    }
```

```
    LinearFit(float x1, double y1, float x2, double y2) {
```

```
        m = 0.;
```

```
25
```

```
        b = 0.;
```

```
        if((x1 - x2) != 0) {
```

```
            m = (y1 - y2) / (x1 - x2);
```

```
            b = y1 - m * x1;
```

```
30
```

```
        }
```

```
    }
```

```
float fitX(float x) {  
    return (float) (m * x + b);  
}
```

5

```
float fitY(float y) {  
    if(m != 0) {  
        return (float) ((y - b) / m);  
    }  
    else {  
        return 0;  
    }  
}
```

10

15


```

// //////////////////////////////////////
// Determines the Peak and Cycle for the second derivative. It takes 3
// points (x,y pairs) and fits a line of the 2nd order through all three
// points. peak(y) is optic and cycle(x) is the PCR Cycle number.
5 // //////////////////////////////////////
class PeakFinder {

    float peak;
    float cycle;
10 double d0, d1, d2, d3;
    double r1, r2, r3;

    PeakFinder () {};

15 PeakFinder(float x1, float y1, float x2, float y2, float x3, float y3) {
    d0 = det((x1 * x1), x1, 1, (x2 * x2), x2, 1, (x3 * x3), x3, 1);

    d1 = det(y1, x1, 1, y2, x2, 1, y3, x3, 1);

20 d2 = det((x1 * x1), y1, 1, (x2 * x2), y2, 1, (x3 * x3), y3, 1);

    d3 = det((x1 * x1), x1, y1, (x2 * x2), x2, y2, (x3 * x3), x3, y3);

    if(d0 != 0f) {
25 r1 = d1 / d0;
    r2 = d2 / d0;
    r3 = d3 / d0;
    cycle = (float) ((-1 * r2) / (2 * r1));
    peak = (float) (r3 - (r2 * r2) / (4 * r1));
30 }
    else {

```

cycle = 0f;
peak = 0f;
}
}

5

// //////////////////////////////////////

double det(float a11, float a12, float a13, float a21, float a22, float a23,
float a31, float a32, float a33) {

10

return ((a11 * a22 * a33) + (a12 * a23 * a31) + (a13 * a21 * a32) -
(a31 * a22 * a13) - (a32 * a23 * a11) - (a33 * a21 * a12));

}

}

15